Efficiently counting all orbits of graphlets of any order in a graph using autogenerated equations

Ine Melckenbeeck, Pieter Audenaert, Didier Colle and Mario Pickavet

January 22, 2018

Abstract

Motivation: Graphlets are a useful tool to determine a graph's small-scale structure. Finding them is exponentially hard with respect to the number of nodes in each graphlet. Therefore, equations can be used to reduce the size of graphlets that need to be enumerated to calculate the number of each graphlet touching each node. [Hočevar and Demšar, 2014] first introduced such equations, which were derived manually, and an algorithm that uses them, but only graphlets with 4 or 5 nodes can be counted this way.
Results: We present a new algorithm for orbit counting, which is applicable to graphlets of any order. This algorithm uses a tree structure to simplify finding orbits, and stabilisers and symmetry-breaking constraints to ensure correctness. This method gives a significant speedup compared to a brute force counting method, and can count orbits beyond the capacity of other available tools.
Availability: An implementation of the algorithm can be found at https://github.com/biointec/jesse
Contact: Pieter.Audenaert@ugent.be

1 Introduction

Graphs' local structure around each node is often indicative of the graphs' function. [Milo et al., 2002] defined *motifs* as small subgraphs of a larger graph that occur more often than statistically expected. Which motifs appear more is an indication of the type of network that is represented by the graph. To find whether a subgraph is a motif for a certain graph, its frequency in that graph must be determined, as well as its frequency in random graphs. The latter may be done by generating random graphs and counting the number of times the subgraph appears ([Lin et al., 2015]) or estimating its frequency in random graphs ([Ginoza and Mugler, 2010]). Counting subgraphs may be done more quickly using GPUs ([Lin et al., 2015]), but efficient methods for off-the-shelf computers also exist ([Houbraken et al., 2014, Van Parys et al., 2017]).

Because guessing which subgraphs will be motifs for a given graph is not trivial, [Przulj et al., 2004] introduced *graphlets*. These are simply all small *induced* connected subgraphs of a given simple undirected graph, not excluding any of them based on frequency. A simple graph is a graph that has no double edges or self-loops. *Small* typically means the subgraphs contain up to 5 nodes; *induced* means the graph nodes are not allowed to

have edges among themselves, other than those included in the graphlet. As a short notation, graphlets of order n can also be called *n-graphlets*. All graphlets of order 3 to 5 can be seen in Figure 1.

An *automorphism* of a graphlet is an isomorphism of that graphlet to itself. More intuitively, it is a permutation of the nodes that leaves the edge set unchanged. The group of automorphisms of a graphlet G is written as $Aut(G)$. Sets of nodes which are mapped onto each other by any automorphism of a graphlet are called *orbits*. In Figure 1 all nodes in each graphlet are marked in different grayscales according to the orbit to which they belong. As a logical extension of nodes' degree, the number of times a graph node touches a specific orbit can be counted. These numbers are called the node's *graphlet degrees* ([Przulj, 2007]). The

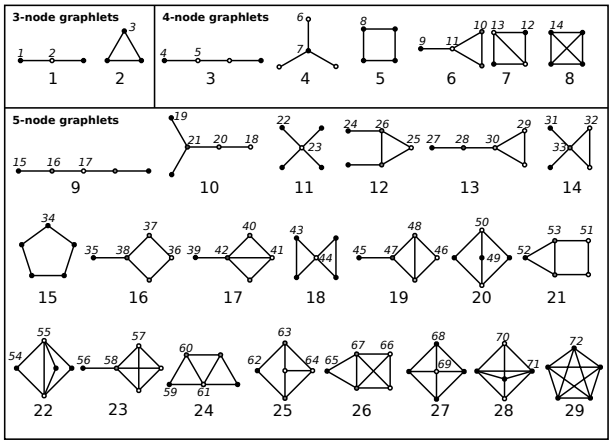

Figure 1: All graphlets of order 3 to 5, with the orbits of each graphlet marked in different grayscales. Each graphlet's number is indicated below it, the number of each orbit is written close to one of its nodes. The two-node graphlet, which is equivalent to a single edge, and its sole orbit are not shown. Figure taken from [Melckenbeeck et al., 2016].

obvious way to find each node's graphlet degrees (using graphlets of order up to n) is to find all graphlets of order up to n. This is, however, a hard problem which scales badly with both n and the number of different graphlets of order n, as well as the graph's order and size. The ways of circumventing this problem include statistical sampling of graphlets ([Rahman et al., 2014]) and using equations to calculate the number of graphlets touching each node ([Hočevar and Demšar, 2014]). This last method allows one to calculate each node's graphlet degrees of order up to n by only enumerating graphlets of order $n-1$. For each graphlet degree of order n, a linear equation is composed relating that graphlet degree to other graphlet degrees and the number of common neighbours of a smaller graphlet. This technique was implemented in the tool ORCA ([Hočevar and Demšar, 2014]), which unfortunately is not scalable; it can only compute the graphlet degrees of 4-graphlets and 5-graphlets. More information about counting graphlet degrees using equations can be found in section 2.4.

The main bottleneck to make the ORCA algorithm scalable is the fact that its equations were composed manually. For each order of graphlets, a whole new set of equations is needed, which gets prohibitively hard for

graphlets containing more than 5 nodes. In earlier work ([Melckenbeeck et al., 2016]) we described a way of automatically generating the needed equations. Here, we will introduce an algorithm that uses them to count graphlets of any size. This pushes the boundary of graphlet degree distributions to larger graphlets than was feasible before.

2 Methods

2.1 Orbit representatives

Graphs can be represented in many different ways. They can be represented as a set of nodes and a set of edges, an incidence matrix, incidence lists for each node, etcetera. In this paper, we only consider simple (i.e. without multiple edges or self-loops), undirected graphs, defined as

$$G = (V, E),$$

where V is the collection of nodes and E is the collection of edges of G, so that

$$E \subseteq \{\{v, w\} | v, w \in V \wedge v \neq w\}.$$

The node set and edge set of a graph are respectively noted as $V(G)$ and $E(G)$, so that

$$G \equiv (V(G), E(G)).$$

Omitting the explicit set of nodes, we choose to represent a graph as the set of its edges, in which each edge is denoted by the labels of the two nodes it connects.

Graphlets are defined as small connected graphs. Just like the large graph, they must be simple and undirected. For small sizes, they can be exhaustively enumerated (see Figure 1). A node in a large graph is said to *touch* a graphlet if the large graph has that graphlet as an induced subgraph with that node included. An induced subgraph of a graph G is a subgraph consisting of a subset of $V(G)$ and precisely all edges among them that are present in G. The nodes of a graphlet are always implicitly labeled v_0 to v_{n-1}, an edge is saved as the set of its nodes as before.

Relabeling the nodes does not change the structure of the graphlet, but it might change the edge list. As such, each graphlet has a number of equivalent representations. In all following examples, *graphlet 18* will be used, along with its *orbit 43* (the numbering from Figure 1 is used). For instance, the edge lists $\{\{0,1\},\{0,2\},\{0,3\},\{0,4\},\{1,2\},\{3,4\}\}$ and $\{\{0,1\},\{0,2\},\{1,2\},\{2,3\},\{2,4\},\{3,4\}\}$ are two different representations of graphlet 18. There are $n!$ possible permutations of the n nodes of a graphlet, $|Aut|$ of them giving the original set of edges, as can be seen in the first two panels of Figure 2. Therefore, there are $\frac{n!}{|Aut|}$ possible edge sets for the same graphlet. Those of graphlet 18 can be seen in both columns of table 1. If all of these edge sets are saved for each graphlet, it can easily be checked whether two different representations do, in fact, represent the same graphlet or not.

The *orbit* of a node v in a graphlet is the set of nodes that can be mapped onto v by some automorphism of the graphlet. Formally, the orbit of node v in graphlet G is defined as:

$$\mathrm{Orb}(G, v) = \{w \in V | \exists \sigma \in \mathrm{Aut}(G) : \sigma(v) = w\}.$$

Orbits are assigned a unique numbering, spanning over all graphlets, so that the notation Orb_j denotes a unique orbit in one unique graphlet (see

Figure 1 for orbit 1 to 72). A node v is said to touch an orbit Orb_j if the graph has Orb_j's graphlet as an induced subgraph, with $v \in \text{Orb}_j$.

If one node of a graphlet is marked and excluded from the permutation, only $(n-1)!$ possible permutations are left. Without loss of generality, this node can always be called node 0. Graphlets like this are called *orbit representatives* ([Melckenbeeck et al., 2016]) and are formally defined as

$$\Omega = (V, E, u)$$

with

$$u \in V \wedge E \subseteq \{\{v, w\} | v, w \in V \wedge v \neq w\},$$

again, so that

$$\Omega \equiv (V(\Omega), E(\Omega), u(\Omega)).$$

Any allowed permutation σ will map the nodes in V to each other, but u will remain unchanged. All n nodes of a n-graphlet can be marked; doing so divides the $n!$ permutations of the original graphlet in n equally sized groups. Marking different nodes from the same orbit will result in groups of permutations that result in the same sets of edge sets, however, marking nodes from different orbits will not. Orbit representatives are assigned the same numbering as orbits (see Figure 1), so that the notation Ω_i denotes the unique orbit representative that corresponds with orbit i.

The third and sixth panel of Figure 2 illustrate how the edge sets of graphlet 18 are split into two groups this way, resulting in different edge sets when an outside node is marked (orbit 43, panel 3) and when the middle node is marked (orbit 44, panel 6). The same result can be seen in edge notation in Table 1.

Table 1: All different edge sets for graphlet 18. They can be split into the edge sets for orbit representative 43 and orbit representative 44.

Edge sets for graphlet 18	
Orbit representative 43	Orbit representative 44
01 02 12 13 14 34	01 02 03 04 12 34
01 02 12 23 24 34	01 02 03 04 13 24
01 03 12 13 14 24	01 02 03 04 14 23
01 03 13 23 24 34	
01 04 12 13 14 23	
01 04 14 23 24 34	
02 03 12 14 23 24	
02 03 13 14 23 34	
02 04 12 13 23 24	
02 04 13 14 24 34	
03 04 12 13 23 34	
03 04 12 14 24 34	

Within each orbit representative, new orbits can be defined using the same definition as orbits in graphlets. Sets of nodes which are mapped onto each other by the remaining automorphisms are said to be *sub-orbits*. Each of these sub-orbits is a subset of a single orbit in the original graphlet; the marked node will always be the sole node in its sub-orbit. Different orbit representatives from the same graphlet can have a different number of sub-orbits; for instance orbit representative 43 has 4 sub-orbits (see panel 5 in Figure 2), while orbit representative 44 has 2 sub-orbits (see panel 8).

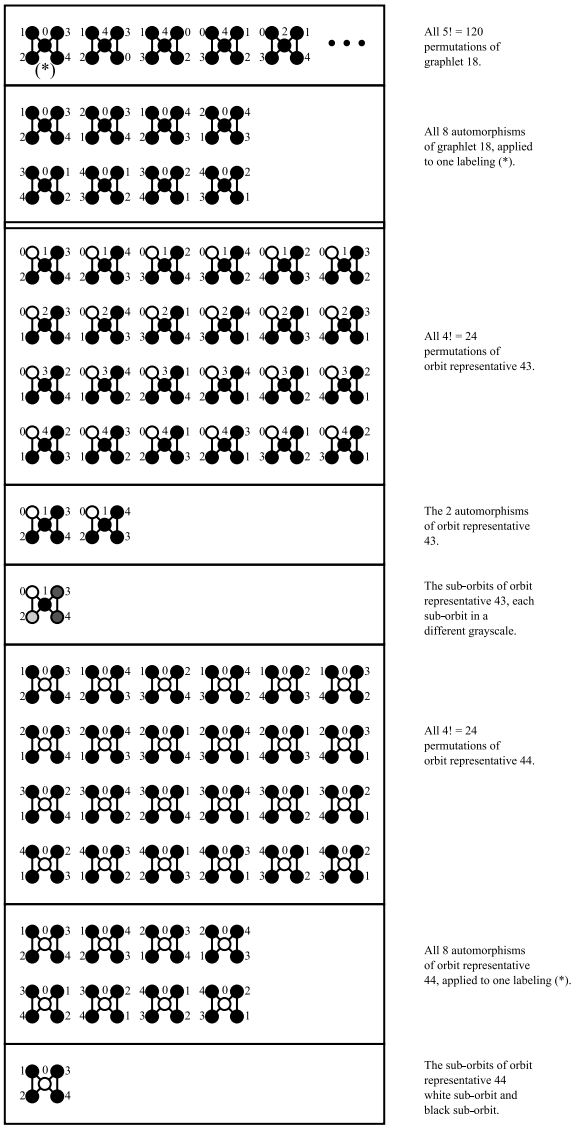

Figure 2: The permutations and automorphisms of graphlet 18 and orbit representatives 43 and 44. In the orbit representatives, the marked node is coloured white.

2.2 Orbit trees

To allow for efficient enumeration of graphlets of any size, we introduce orbit trees. The reason for their particular structure will become clear in section 2.3. These orbit trees will be introduced in three steps.

2.2.1 An introductory orbit graph

All orbit representatives can be constructed by a series of atomary additions of nodes and edges to the orbit representative containing only the marked node 0, which will be called the *minimal orbit representative*. This way, orbit representatives themselves can be seen as the nodes in a directed graph, and the addition of nodes and edges can be seen as the directed arcs in that graph (see Figure 3). Because orbit representatives, like graphlets, need to be connected graphs, a new node is always added with an edge to one of the other nodes of the orbit representative.

Let $\mathcal{V}_{or}$ be the set of all orbit representatives on n or less nodes. These are the nodes of the *orbit graph*, which are connected by two types of arcs. One type of arc represents adding an edge between two pre-existing nodes of the orbit representative. These arcs, represented as dashed arrows in Figure 3, can be defined as:

$$E_e = \{(\Omega_i, \Omega_j) | \exists e \notin E(\Omega_i) : \Omega_j \cong (V(\Omega_i), E(\Omega_i) \cup \{e\}, u(\Omega_i))\},$$

that is, two orbit representatives Ω_i and Ω_j are connected by a dashed arrow if and only if Ω_j is isomorphic to Ω_i with one additional edge (the $\cong$ symbol denotes isomorphism).

The other type of arc represents the addition of a node, together with an edge, to the orbit representative. This set of arcs, represented as full arcs in Figure 3, is defined by :

$$E_{ne} = \{(\Omega_i, \Omega_j) | \exists v \in V(\Omega_i), w \notin V(\Omega_i) : \Omega_j \cong (V(\Omega_i) \cup \{w\}, E(\Omega_i) \cup \{\{v, w\}\}, u(\Omega_i))\}, \quad (1)$$

that is, two orbit representatives Ω_i and Ω_j are connected by a full arrow if and only if Ω_j is isomorphic to Ω_i with one additional node and an edge connecting it to a node $v \in V(\Omega_i)$.

With the previous definitions, the orbit graph is defined as

$$G_{OG} = (\mathcal{V}_{or}, E_e \cup E_{ne}).$$

The orbit graph containing orbit representatives with up to 4 nodes is depicted in Figure 3. This graph serves as the basis for the next step. All figures in this section (figure 2 to 6) are generated automatically by Java implementations of the described algorithms.

2.2.2 An extended orbit graph

In the next step, we need to pay attention to the order in which nodes are added to the orbit representative. Without loss of generality, only edges connecting the newest node to another node need to be added to any orbit representative. As every node has been the most recent node at some point, each possible edge will be added.

A new type of edge within an orbit representative will be introduced now. It represents an edge that may still be present or absent in the current orbit representative, but will be decided later on to be absent or

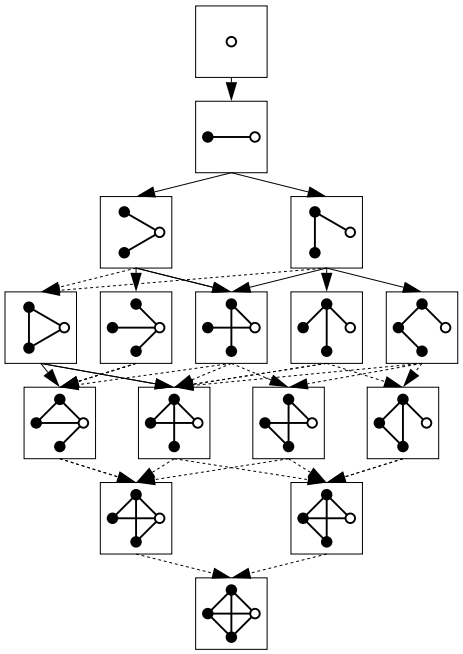

Figure 3: The orbit graph G_{OG} containing all graphlets of order up to 4. The white node is the marked node of the orbit representatives. Full arcs represent the addition of a node to the orbit representative (with its connecting edge), dashed arcs represent the addition of an edge.

present. We will call this type of edge *gray* edges, to indicate they are either present (black) or absent (white).

Every time a new node is added to an orbit representative (arcs in E_{ne}), it is now added with a (black) edge to one present node as before, but it will have gray edges to all other nodes. One by one, gray edges are changed to be either present or absent, in increasing order of the index of the nodes involved. To illustrate this, Figure 4 shows the orbit graph created this way, with the orbit representatives shown as matrices, while Figure 5 shows the exact same orbit graph with drawn orbit representatives.

Graphlets can be represented in matrix notation, in which the matrix element at position i, j represents the edge between node i and node j. In undirected graphs, like orbit representatives, these matrices are symmetrical, so that they can be represented as upper triangular matrices without losing any information. The possible values for each matrix element are 0 (no edge), 1 (an edge) or g (gray edge).

In these figures, it can be seen that orbit representatives are actually split into 2 groups: those with and without gray edges. In analogy with their outgoing edges, these two sets of nodes can be called V_e and V_{ne}. Orbit representatives in V_{ne}, without any gray edges, only have children with gray edges. More specifically, they all have every possible child that results from adding a new node with one black edge and all other possible gray edges. Therefore, the last column in the matrix has one 1, all other values in that column are g.

Each orbit representative in V_e, with at least one gray edge, has two children: the orbit representative in which the topmost gray edge in the matrix representation is absent, and the orbit representative in which that edge is present; that is: value 0 or 1 in the matrix, respectively. The order from topmost to bottommost gray edge in the matrix representation, or equivalently in increasing order of index of the node involved, was chosen arbitrarily; any other order would work equally well.

2.2.3 An orbit tree

In the last step, we will remove superfluous edges from the orbit graph. In the orbit graph, there may be multiple directed paths from the minimal orbit representative to some other orbit representative. These different paths represent different ways to construct the orbit representative, but only one is needed for each orbit representative. Therefore, a rooted, directed spanning tree of the orbit graph from the previous section 2.2.2 is constructed with the minimal orbit representative as root. Practically, this means if any orbit representative has multiple incoming arcs in the orbit graph, all but one of them are removed. When this is done with the graph from Figure 5, Figure 6 is created. If this process removes all outgoing arcs from an orbit tree node in V_e, that orbit tree node is also removed. The end result is an *orbit tree* in which there is exactly one orbit tree node in V_{ne} for each orbit representative; moreover, every leaf of the orbit tree will be of this type.

Pseudocode for the algorithm to build this orbit tree is shown in Algorithm 1.

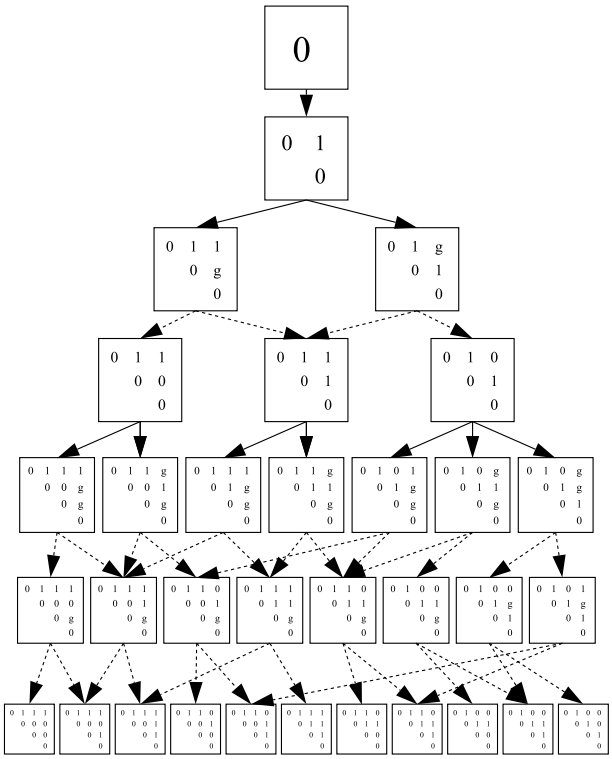

Figure 4: The orbit graph with gray edges, shown as matrices. Every node is added with one certain edge (1), and all possible other edges as uncertain or gray (g). The children of an orbit representative with at least one gray edge are the two orbit representatives in which that edge is present or absent, respectively. The first row of the matrix corresponds with the marked node. Full arcs represent the addition of a node to the orbit representative (with its connecting edge), dashed arcs represent the addition of an edge.

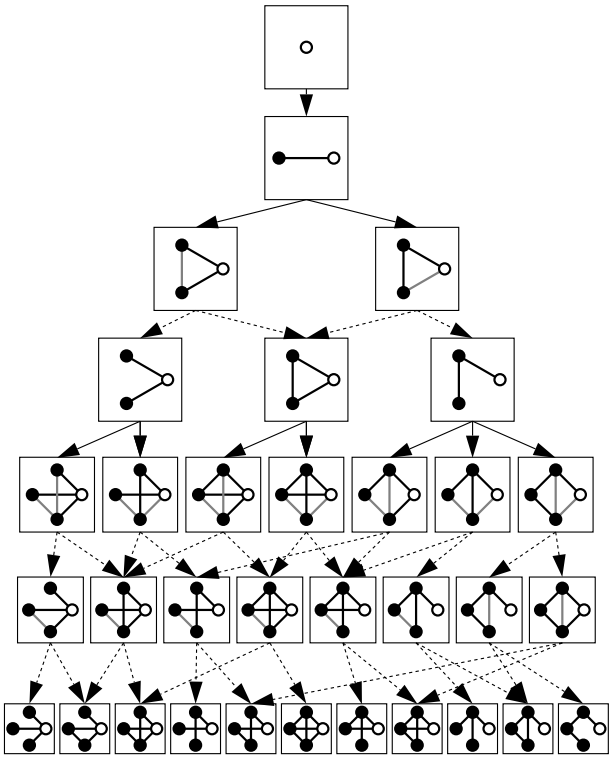

Figure 5: The orbit graph with gray edges, shown as drawings. The white node is the marked node of the orbit representatives. Gray edges are edges that still may or may not be present. Full arcs represent the addition of a node to the orbit representative (with its connecting edge), dashed arcs represent the addition of an edge.

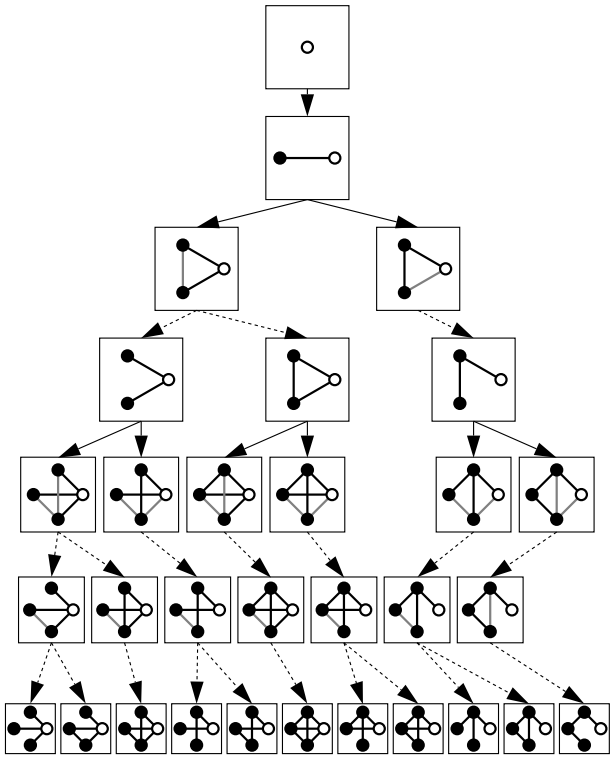

Figure 6: The orbit tree containing all graphlets of order up to 4. The white node is the marked node of the orbit representatives. Full arcs represent the addition of a node and its connecting edge to the orbit representative, dashed arcs represent the addition of an edge.

Algorithm 1 The algorithm to build the tree

function BUILDTREE(maximal graphlet size n)
    root $\leftarrow$ minimal orbit representative
    $V_{ne} = \{$ root$\}$
    $V_e = \emptyset$
    for $i = 2 \rightarrow n$ **do**
        while $V_{ne} \neq \emptyset$ **do**
            remove the first element of V_{ne}, call it a
            a.children $\leftarrow$ all possible gray edge orbit representatives made by adding a node with its connecting edge to a
            $V_e \leftarrow V_e \cup a$.children
        end while
        while $V_e \neq \emptyset$ **do**
            remove the first element of V_e, call it a
            $b \leftarrow a$ with first gray edge set to 0
            $c \leftarrow a$ with first gray edge set to 1
            a.children $\leftarrow (b, c)$
            if b contains a gray edge **then**
                $V_e \leftarrow V_e \cup a$.children
            else
                $V_{ne} \leftarrow V_{ne} \cup a$.children
            end if
        end while
    end for
    create a tree from this graph rooted in root, e.g. with Prim's algorithm
    prune all branches of the tree ending in a gray edge orbit representative
end function

2.3 Finding graphlets with the tree

As the orbit tree represents the way that orbit representatives are constructed, it can be used to search and find those orbit representatives within a graph. As each orbit representative appears exactly one time in the orbit tree as a node in V_{ne}, i.e. an orbit representative without any gray edge, these nodes will be used to actually count the orbits. Simply speaking, finding which orbits touch a node v comes down to finding each connected set of n nodes containing v and identifying which orbit representative they form (with v marked).

To do so, the orbit tree can be used. Assume we are trying to find each graphlet and orbit of order up to n touching node x. This can be done via a straightforward matching algorithm. First, an orbit tree containing the graphlets of order up to n is built (using algorithm 1, this only needs to be done once), then algorithm 2 is called with the following three arguments: a mapping m containing only node x, the tree's root, and the graph G itself. The mapping maps the nodes from the current orbit representative to the graph and will be filled while searching for instances of orbit representatives. The algorithm walks through the tree: whenever it encounters an orbit tree node from V_{ne}, a new node w and its connecting edge to node v will be added to the orbit representative (see equation 1). Thus, the algorithm will call itself recursively for all children of the orbit tree node and all possible graph nodes y that can serve as this new node w. In contrast, when it encounters an orbit tree node from V_e, the algorithm will check whether its first gray edge is present in the graph or not, then recursively call itself for the appropriate child.

Algorithm 2 Finding orbit representatives in a graph using the tree

```
function findOrbitRepresentatives(mapping m, tree node t, graph G)
    if t ∈ V_ne then
        orbit representative found, process m as an instance of t
        for all tn child of t do
            {v, w} ← the newly added black edge in tn
            for all y ∈ V(G) such that {m(v), y} ∈ E(G) do
                findOrbitRepresentatives(m ∪ [w → y], tn, G)
            end for
        end for
    else // that is: t ∈ V_e
        {v, w} ← the first gray edge in t
        if {m(v), m(w)} ∈ E(G) then
            findOrbitRepresentatives(m, t.child_with_edge, G)
        else
            findOrbitRepresentatives(m, t.child_without_edge, G)
        end if
    end if
end function
```

This method will, however, find orbit representatives multiple times. If the orbit representative has more than one automorphism, multiple equivalent mappings of graph nodes to orbit representative nodes can happen. For example, the orbit representative in figure 7 can be mapped to the graph in 2 different ways if the orbit representative's marked node is fixed, and these two mappings will be returned by algorithm 2. The

number of times this can happen is exactly the number of automorphisms of the orbit representative. For all intents and purposes of this paper, we are only interested in one single instance of an orbit representative on each given set of nodes. Therefore, the number of instances found this way can be divided by the number of automorphisms to get the real number of times the orbit representative touches the given node.

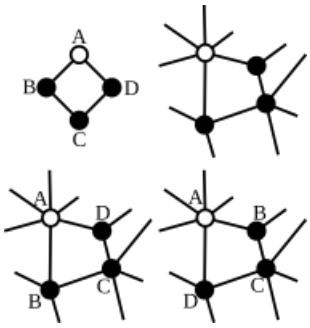

Figure 7: Top row: an orbit representative and part of a graph, with 4 nodes. Bottom row: two different correct mappings of the orbit representative to the graph.

However, when the instances of the orbit representatives are needed, not just the number, it is not possible to use algorithm 2 straight away: it would still return both mappings in the bottom row of figure 7, while we only need one of them. It would be possible to discard double instances, but looking up each instance when it is found, is not time-efficient. Therefore, we need to impose symmetry-breaking constraints on the nodes on which the orbit representative is mapped. These can be used to unambiguously single out a unique mapping of an orbit representative to a set of nodes from a graph.

The method used to find these symmetry-breaking constraints is the one that is used in [Houbraken et al., 2014], for which a few definitions are needed. The stabiliser i_P of an element i for a permutation group P is defined as the set of permutations within that group that leave that element unchanged. Formally:

$$i_P = \{\sigma \in P | \sigma(i) = i\}$$

It is also possible to stabilise multiple nodes one after another. A *stabiliser chain* is a sequence of stabilisers, each explicitely stabilising one extra node in addition to those stabilised by the previous stabiliser in the chain. The chain can be defined as:

$$P_i = \begin{cases} P, & i = 0, \\ i_{P_{i-1}}, & i > 0. \end{cases}$$

As we are working with orbit representatives, the permutation group used will be the automorphism group of an orbit representative. The permuted elements are the orbit representative's nodes, and they will be stabilised in the order they were added to the orbit representative. Figure 8 shows an example of a stabiliser chain in orbit representative 44. As can be seen in figure 8, stabilising a node can automatically stabilise some other nodes, while at the same time leaving still other nodes unstabilised.

For example, node 2 is stabilised as soon as node 1 is stabilised, but nodes 3 and 4 are not.

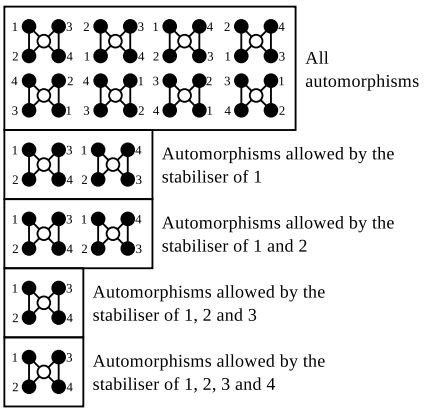

Figure 8: An example of the stabiliser chain of orbit representative 44. In each row, all labelings of the orbit representatives allowed by the stabiliser are depicted.

To be able to use these stabilisers to avoid getting double instances as discussed above, we need an additional definition. First of all, we assume each graph node $v \in V(G)$ has a unique number, called its index, which runs from 1 to $|V(G)|$. The function $I(v) : V(G) \to \mathbb{N}$ returns this index for every node v. These indices will be used to impose constraints that will allow only one instance of an orbit representative for each set of nodes.

These so-called symmetry-breaking constraints on the node indices can be derived as follows. Just before a node v gets stabilised explicitly, some other nodes might still be interchangeable with v. Therefore, there needs to be a constraint preventing these nodes to be swapped with v during the orbit representative search. Given an orbit representative Ω, the set of constraints $C(\Omega)$ imposed on nodes $v \in V(\Omega)$ and $w \in V(\Omega)$ is formally defined as:

$$C(\Omega) = \{(v, w) | \exists \sigma \in Aut(\Omega)_{v-1} : \sigma(v) = w\}$$

For every $(v, w) \in C(\Omega)$, we now impose that $I(m(v)) < I(m(w))$ with m the mapping as described above.

As an illustration, the constraints for the orbit representative 44 can be derived from figure 8. With no node stabilised, node 1 can be interchanged with node 2. Now assume node 1 is mapped to graph node $m(1) = x \in V(G)$ and node 2 to $m(2) = y \in V(G)$, then we have the constraint $I(x) < I(y)$. However, node 3 and 4 can still be interchanged with each other, even after nodes 1 and 2 are stabilised, therefore we need a second constraint $I(m(3)) < I(m(4))$.

During the execution of algorithm 2, these constraints can be enforced easily, discarding double instances, thus solving the problem of multiple orbit representatives.

2.4 Counting graphlets using equations

[Hočevar and Demšar, 2014] showed that the numbers of all different n-orbits touching each node in a graph can be related to each other with a set

of linear equations. These equations relate the graphlet degrees of graphlet order n to the number of common neighbours of the nodes of graphlets of order $n-1$. In effect, they make it possible to find the graphlet degrees of order n by only enumerating the graphlets of order $n-1$. The equations can be generated automatically ([Melckenbeeck et al., 2016]) and use the following additional definitions: $P_i(x,\ldots)$ which is a function that selects instances of orbit representative i with x as the marked node, $o_i(x)$ which is graphlet degree i of node x, and $c(x_1,\ldots,x_n)$ which is the number of common neighbours of x_1 to x_n. Full formal definitions and sets of equations can be found in the papers cited above, but for illustrative purposes we provide the following sample equation:

$$\begin{aligned} 2o_{71}(x) + 12o_{72}(x) = \sum_{a,b,c:P_{14}(x,a,b,c)} & (c(x,a,b)-1)+ \\ & (c(x,a,c)-1)+ \\ & (c(x,b,c)-1) \end{aligned}$$

Typically, there are many more equations than needed to solve the orbit counting problem, so a subset of them needs to be selected. Moreover, the equations can be solved sequentially in descending order of orbit number in the left-hand side.

2.4.1 Generating equations

In previous work ([Melckenbeeck et al., 2016]), we showed in full detail how the equations are generated. While this is not the main scope of this paper, a short explanation follows.

Equations arise every time a new orbit representative is constructed by adding a node to a smaller sized orbit representative, along with a number of connecting edges. For each orbit representative, it is sufficient to have a single equation so in total we need as many equations as there are orbit representatives of order n. Solving these equations one by one results in the orbit degrees we are looking for. All possible equations can be generated by exploiting all possible structural relations between orbit representatives of order $n-1$ and order n. This can be done in a combinatorial way: by finding all these relations, we can link the presence of an orbit representative of order n to the presence of other orbit representatives of order $n-1$. Thus, if we have enumerated all orbit representatives of order $n-1$, we can calculate the number of orbit representatives of order n.

In general, many more equations are generated than there are needed to calculate the orbit degrees, because one orbit representative can have many relations to many other orbit representatives. However, these equations are linearly dependent on each other, so that it is sufficient to generate only a small selection of equations.

Algorithms, examples and illustrations of equation generation, together with a fully detailed explanation, can be found in [Melckenbeeck et al., 2016].

3 Implementation

The list of equations is normally ordered by the lowest numbered orbit representative in the left-hand side of each equation. When using the equations, it is more useful to first order them by the orbit representative

over which the sum in the right-hand side is made. Then, during algorithm 2, only the right-hand side of each equation is needed, plus a way to identify each equation. Therefore, during the algorithm, each equation in a group is reduced to just the number of the equation, a list of lists of integers indicating the groups of common neighbours in the right-hand side and an integer indicating the negative term in the right-hand side.

Before the search for orbit representatives, the value of the right-hand side of each equation is set to be zero. When an orbit representative is found with the tree, the corresponding group of equations is accessed. For each equation, the number of common neighbours of the needed nodes is retrieved and the correction term is subtracted. This number is added to the value of the right-hand side of the equation. After all orbit representatives have been found, all equations are ordered by lowest number of the graphlets in the left-hand side and solved in descending order of that number.

4 Results and discussion

4.1 The Jesse tool

At https://github.com/biointec/jesse, a Java implementation of the algorithm can be found. This implementation bundles all of the tools needed to count all orbits of graphlets of any order in a graph. It will generate a unique numbering for all graphlets up to the given order and their orbits, build an orbit tree to find all orbit representatives of orders smaller than the given order, generate new equations and use all of the aforementioned steps to determine, for each node of the graph, the number of all orbits touching that node. The result of each step may be saved to be used in later runs.

There are two ways to run the tool: with or without arguments. When run without arguments, all options are available: the graphlets, tree and equations can be selected individually to be generated and potentially saved, or read from file. The program will ask to specify these parameters and file names in the command line. With arguments, all of the needed data are read from files which are specified in the arguments.

4.2 Application to networks

In order to assess the quality of our algorithm, we performed the following experiments.

First we applied the Jesse tool on the graph [Hočevar and Demšar, 2014] used. This graph consists of 100 nodes and 1000 edges. Three parts of the algorithm were timed: counting the common neighbours of sets of nodes; building a tree of order $n-1$ together with an interpreter for the tree and a system of equation to count all orbits of order n; and actually counting the orbits of order n with this tree. These timings were benchmarked against a brute force approach counting all orbits of order n. The *speedup factor* (f_s in formulae) in this table is the brute force time divided by the time of the Jesse approach. The orbits of graphlets containing 3 to 6 nodes were counted. Each run was repeated 20 times to average the effect of Java's dynamic memory allocation. Results can be seen in table 2. The use of equations effects a significant speedup in the graphlet counting for graphlet order of 4 or greater. Graphlets of 3 nodes are counted more slowly due to the overhead needed to use equations.

Table 2: Running times of the Jesse tool for different graphlet orders. The graph used is the graph included with ORCA: it has 100 nodes and 1000 edges.

Number of nodes	Time (s)					Speedup factor
	Common neighbours	Tree & interpreter	Counting orbits	Total	Brute force	
3	0.0001	0.0067	0.0169	0.0237	0.0120	0.5060
4	0.0157	0.0119	0.0606	0.0882	0.2170	2.4598
5	0.0647	0.0564	1.5976	1.7187	5.2203	3.0373
6	0.5524	3.5135	90.3801	94.4460	178.7752	1.8929

Graphlets of order greater than 3 can be counted more efficiently with equations than they can be found without equations. Graphlets of order 4 and 5 are counted less quickly than the ORCA algorithm does ($< 0.01s$ and $0.08s$ respectively, [Hočevar and Demšar, 2014]), but Jesse is not specifically optimised for these cases, while ORCA is. Instead, Jesse can also count larger graphlets, from order 6 onwards, which ORCA completely fails to do.

The properties of the explored graph also affect Jesse's running time. The relation between the number of edges in a graph and the efficiency of Jesse can be seen in table 3. Graphs used were Erdös-Rényi ([Erdos and Renyi, 1959]), Barabási-Albert or scale-free ([Barabási et al., 1999]) and Geometric ([Przulj, 2007]) random graphs with 100 to 200 nodes and a variable density parameter. For each experiment, a new graph was generated, after which the orbits of all graphlets on 5 or less, and 6 or less nodes were counted with Jesse and the brute force approach. Each experiment was repeated 20 times, with the exception of those taking longer than an hour. It can clearly be seen that the denser the graph, the higher Jesse's speedup; this holds true in each type and order of graph that was investigated. Indeed, the more edges there are in a graph, the more graphlets there need to be counted. And because Jesse is faster in counting graphlets than brute force counting, its speedup factor will increase with the graph's density.

Figure 9 shows some graphlet degrees in different types of network. The model network is the kinase and phosphatase physical interaction network in yeast described in [Breitkreutz et al., 2010]. This network has 887 nodes and 1844 edges. Therefore, it was compared with ER networks with 887 nodes and 1844 edges ([Erdos and Renyi, 1959]); BA (scale-free) networks with 887 nodes and 2 edges per node, resulting in 1770 edges ([Barabási et al., 1999]); and geometric networks with 887 nodes, 3 dimensions and a neighborhood radius of 0.11 (resulting in on average 1921 edges) ([Przulj, 2007]). Of each of these types, 100 random graphs were generated and compared to the real life biological interaction network.

5 Conclusion

We presented a new algorithm for orbit counting, which is applicable to graphlets of any order. The algorithm counts the number of times each node touches each orbit of all graphlets of the given order and lower. To this aim, we use a tree structure to streamline finding orbits, and symmetry analysis of the orbit representatives prevents finding double instances. This is the first algorithm to be able to count orbits of any order using equations. It is not as optimised for specific cases as ORCA

Table 3: Running times of the Jesse tool different graph types of different sizes and orders. The density parameter is the number of edges in Erdős-Rényi graphs, the degree of new nodes for Barabási-Albert graphs, and the neighborhood radius in 3D geometric networks.

Graph type	Order	Density Parameter	5-graphlets			6-graphlets		
			Time(s)			Time(s)		
			Brute force	With equations	Speedup factor	Brute force	With equations	Speed fac
Erdős-Rényi	100	800	2.14	1.02	2.10	59.50	42.76	1
		1000	4.60	1.71	2.69	140.80	86.94	1
		1200	8.46	2.64	3.19	275.39	144.43	1
	150	1200	3.72	1.70	2.17	123.63	83.34	1
		1500	8.15	3.14	2.59	316.55	175.35	1
		1800	15.53	5.13	3.02	669.61	323.51	2
	200	1200	5.23	2.41	2.16	193.70	128.89	1
		1500	12.14	4.75	2.55	511.87	288.05	1
		1800	23.66	8.31	2.84	1145.71	558.85	2
Barabási-Albert	100	8	3.12	1.04	2.98	97.03	44.74	2
		10	5.48	1.52	3.59	184.30	75.12	2
		12	8.20	2.11	3.87	282.03	108.63	2
	150	8	7.81	2.30	3.38	333.94	129.97	2
		10	13.97	3.66	3.81	628.29	227.25	2
		12	22.17	5.31	4.17	1020.26	359.33	2
	200	8	14.20	3.94	3.60	744.47	256.19	2
		10	25.79	6.32	4.07	1726.71	546.83	3
		12	42.57	9.55	4.45	2643.26	837.18	3
Geometric (3D)	100	0.37	0.57	0.29	1.98	13.57	10.63	1
		0.46	5.04	1.35	3.73	142.28	58.70	2
		0.53	13.93	2.88	4.82	482.91	134.54	3
	150	0.37	4.69	1.55	3.00	128.29	63.64	2
		0.46	38.59	7.67	5.03	1490.92	429.49	3
		0.53	110.41	7.26	6.40	7603.49	1482.40	5
	200	0.37	18.39	4.97	3.70	697.93	267.14	2
		0.46	159.63	26.05	6.13	12252.67	2631.29	4
		0.53	456.18	58.15	7.84	Timeout		

is, but it can handle graphlets of higher order and is much more widely applicable.

The Jesse tool implements the algorithm and provides a best-case speedup of nearly up to a factor 8, getting more efficient compared to a brute force approach the more graphlets there are.

Acknowledgements

Funding

This work has been supported by Ghent University and imec.

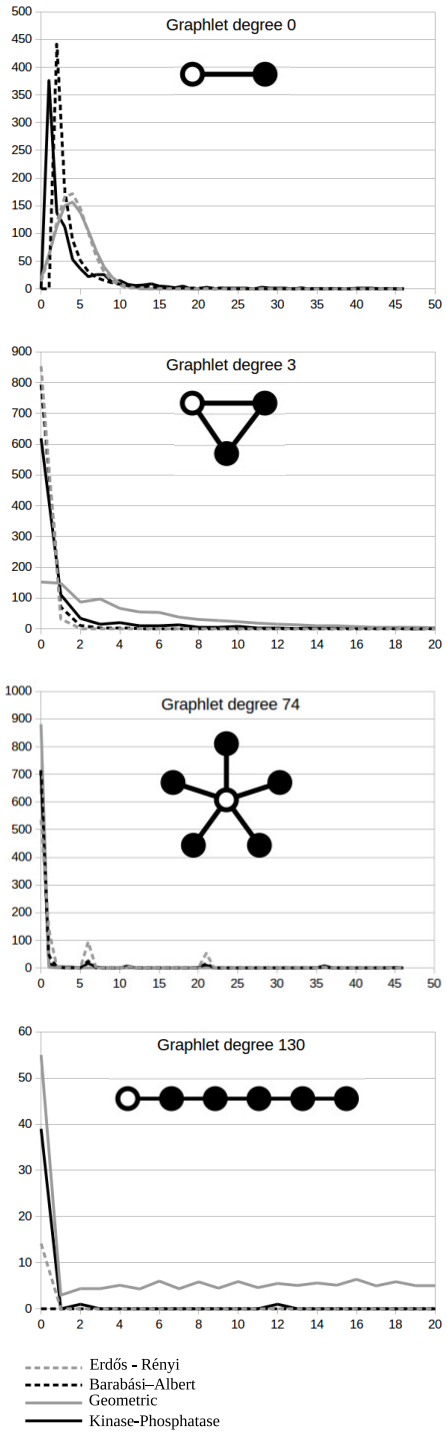

Figure 9: The Graphlet Degree Distribution for a few different orbits in different random graphs. On the x-axes, the number of times a node touches a graphlet can be found; on the y-axes, the number of nodes touching this number of instances of this orbit. Most graphlets containing loops are rare.

References

[Barabási et al., 1999] Barabási, A., Albert, R., and Jeong, H. (1999). Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications*, 272:173–187.

[Breitkreutz et al., 2010] Breitkreutz, A., Choi, H., Sharom, J. R., Boucher, L., Larsen, B., Lin, Z.-y., Breitkreutz, B.-j., Stark, C., Ahn, J., Dewar-darch, D., Reguly, T., Tang, X., Qin, Z. S., Pawson, T., Gingras, A.-c., Nesvizhskii, A. I., and Tyers, M. (2010). A Global Protein Kinas and Phosphatase Interaction Network in Yeast. *Science*, 328(5981):1043–1046.

[Erdos and Renyi, 1959] Erdos, P. and Renyi, A. (1959). On random graphs. *Publicationes Mathematicae.*, (6):290–297.

[Ginoza and Mugler, 2010] Ginoza, R. and Mugler, A. (2010). Network motifs come in sets: Correlations in the randomization process. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 82(1):1–6.

[Hočevar and Demšar, 2014] Hočevar, T. and Demšar, J. (2014). A combinatorial approach to graphlet counting. *Bioinformatics (Oxford, England)*, 30(4):559–65.

[Houbraken et al., 2014] Houbraken, M., Demeyer, S., Michoel, T., Audenaert, P., Colle, D., and Pickavet, M. (2014). The Index-based Subgraph Matching Algorithm with General Symmetries (ISMAGS): exploiting symmetry for faster subgraph enumeration. *PloS one*, 9(5):e97896.

[Lin et al., 2015] Lin, W., Xiao, X., Xie, X., and Li, X. L. (2015). Network motif discovery: A GPU approach. In *2015 IEEE 31st International Conference on Data Engineering*, pages 831–842.

[Melckenbeeck et al., 2016] Melckenbeeck, I., Audenaert, P., Michoel, T., Colle, D., and Pickavet, M. (2016). An Algorithm to Automatically Generate the Combinatorial Orbit Counting Equations. *PLoS ONE*, 11(1):1–19.

[Milo et al., 2002] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. (2002). Network motifs: simple building blocks of complex networks. *Science (New York, N.Y.)*, 298(5594):824–7.

[Przulj, 2007] Przulj, N. (2007). Biological network comparison using graphlet degree distribution. *Bioinformatics (Oxford, England)*, 23(2):e177–83.

[Przulj et al., 2004] Przulj, N., Corneil, D. G., and Jurisica, I. (2004). Modeling interactome: scale-free or geometric? *Bioinformatics (Oxford, England)*, 20(18):3508–15.

[Rahman et al., 2014] Rahman, M. M., Alam, M., Rahman, M. M., Al, M., Bhuiyan, M. A., Rahman, M. M., and Hasan, M. A. (2014). GUISE: a uniform sampler for constructing frequency histogram of graphlets. *Knowledge and Information Systems*, 38(3):511–536.

[Van Parys et al., 2017] Van Parys, T., Melckenbeeck, I., Houbraken, M., Audenaert, P., Colle, D., Pickavet, M., Demeester, P., de Peer, Y., Parys, T. V., Melckenbeeck, I., Houbraken, M., Audenaert, P., Colle, D., Pickavet, M., Demeester, P., and Peer, Y. V. D. (2017). A Cytoscape app for motif enumeration with ISMAGS. *Bioinformatics*, 33(3):461.